A Toolset for Modelling and Verification of GALS Systems

S. Ramesh, Sampada Sonalkar, Vijay D'silva,
Naveen Chandra R., and B. Vijayalakshmi

Center for Formal Design and Verification of Software
Department of Computer Science and Engineering, IIT Bombay
ramesh@cse.iitb.ac.in

1 Introduction

We present a toolset for design and verification of Globally Asynchronous Locally Synchronous(GALS) systems. Such systems consist of a network of reactive nodes which have independent clocks and I/O interfaces, and communicate using complex synchronisation mechanisms. GALS systems are gaining prevalence in avionics, embedded systems, and VLSI design. These systems are difficult to design and verify due to the concurrency and complex interaction involved.

The toolset is based on a visual formal language called Communicating Reactive State Machines(CRSM)[6], which builds upon Communicating Reactive Processes[2]. It seamlessly integrates a graphical editor, a simulator and a verification engine. It has several novel aspects in the areas of language design and verification. The semantics of CRSM consolidates ideas from the synchronous languages with classical concurrency constructs. The simulator implements a distributed protocol to incorporate pre-emption with asynchronous communication and supports distributed simulation with context switches. Properties are specified using *distributed observers* and verified using Spin[4]. The verification engine includes a non-trivial translation from CRSM, an open system with GALS semantics, to Promela, a closed system with asynchronous semantics. In addition, Spin has been modified to generate counter examples that can be viewed directly in the simulator.

We have used the tools to model and verify standard pedagogical examples, and for technology transfer in a company. We have found CRSM well suited for providing cycle accurate descriptions of control dominated architectures with multiple clock domains. Industrial case studies include a multi-processor System-on-Chip(SoC) application and a bus protocol. In this paper, we illustrate these tools using a case study. Section 2 introduces underlying theory, Section 3 discusses the tools, implementation issues, and our experience, and Section 4 concludes.

2 Communicating Reactive State Machines

A CRSM is a network of nodes built from communicating boolean Mealy-style automata using constructs for synchronous and asynchronous parallel composi-

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 506–509, 2004.

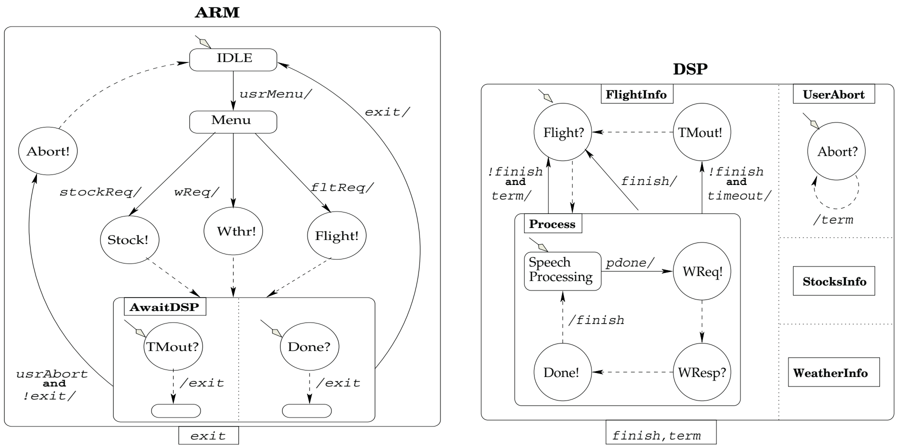

Fig. 1. *ARM* and *DSP* components of Infophone

tion, hierarchy, and signal hiding. The nodes are locally synchronous, execute concurrently, emit signals via synchronous broadcast, and communicate on point-to-point channels using CSP-style rendezvous. A formal description is provided in [9]. We illustrate CRSM using Infophone[1], a speech enabled Java application for information retrieval. It uses an ARM processor to control the user interface, a DSP to process speech commands, and a wireless web interface.

A CRSM description of Infophone is written as $ARM//DSP//WEB$, where $//$ is the operator for asynchronous parallel composition. Figure 1 shows simplified versions of ARM and DSP. The rectangles and circles represent passive and communication states respectively. The dashed arrows from communication states denote transitions taken when communication succeeds and solid arrows, transitions taken when their guards are true; guards describe the status of signals in the environment.

When activated by the signal `usrMenu`, ARM receives the user's request, say `fltReq` and forwards it to DSP on the relevant channel, in this case `Flight`. DSP receives speech commands, sends a request to WEB on `WReq` and notifies ARM when a response is received on `WResp`. A session ends in three ways: successfully, when ARM receives a message on the channel `Done` from DSP, times out, when a `timeout` is issued by DSP if WEB is not responding, and aborts, when the user issues `usrAbort`.

The state `AwaitDSP` in ARM has hierarchy and contains two automata. The transitions leaving `AwaitDSP` allow these automata to complete their ongoing reaction before passivating them, a policy of *weak pre-emption*. The automata `FlightInfo, StockInfo, WeatherInfo` and `UserAbort` in DSP execute in synchronous parallelism, written `FlightInfo`||..||`UserAbort` and interact

[1] Infophone was developed on the Open Multimedia Application Platform(OMAP), a trademark of Texas Instruments.

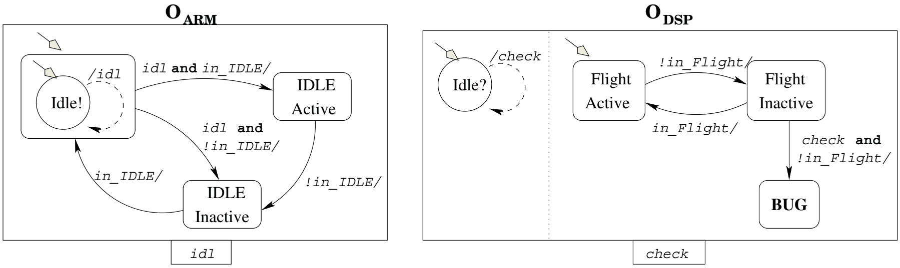

Fig. 2. Distributed Observers: O_{ARM} and O_{DSP}

using local signals such as `exit, term` and `finish`. Nodes are required to be deterministic and constructive[1, 9].

Safety properties of CRSMs are specified using distributed observers. An observer monitors the status of a node, communicates with other observers, and enters a special state `Bug` when a property is violated. Verification involves checking the system $(ARM \| O_{ARM})//(DSP \| O_{DSP})//(WEB \| O_{WEB})$ for the reachability of the state `Bug`. The observers in Figure 2 specify that a session terminated in ARM should be terminated in DSP within its next cycle. The conditional $in\_\langle state \rangle$ holds if the $state$ is active at the end of the reaction. A subtle error occurs when `usrAbort` is issued in ARM and `timeout` in DSP. ARM transits to the state $Abort!$, communicates with the automaton `UserAbort` in DSP and enters `Idle`. DSP consequently enters the state $TMout!$. The next time `fltReq` is issued in ARM, the system will deadlock.

3 Experience and Discussion

We have developed a graphical environment which integrates the design and verification flow described. CRSM models can be built using a graphical editor or a textual language `tCRSM`. The execution sequences can be viewed in the simulator, which performs a *must* and *can* analysis to determine the status of local signals[1] and implements a distributed protocol[7] to address issues due to pre-emption in the presence of communication[8].

Model checking is performed by translating the system to Promela[5, 10]. The Promela code for each node includes a reactive kernel and an environment process, and ensures that the status of signals and states in the system are evaluated correctly. Signal hiding requires *must* and *can* analysis to be incorporated in the Promela code. Spin is invoked automatically with the specification $\Box\neg$`Bug`(*always not* `Bug`) and counter examples generated are translated into traces and displayed in the simulator. Spin has been modified for this purpose.

The tools described have been implemented in approximately 30,000 lines of C and Java code. Our case studies include Infophone and a proprietary bus protocol used by Texas Instruments. The Infophone system with its observers was translated into 890 lines of Promela code with 107 boolean variables, while the bus protocol was translated to 270 lines of Promela code with 31 boolean variables. Boolean variables are required for state and signal encoding, performing analysis, providing observer related primitives and implementing rendezvous. The absence of local signals and asynchronous communication resulted in fewer booleans in the bus protocol code. We observed that CRSM models provided Register Transfer Level(RTL) style structure, yet complete and cycle accurate descriptions of the bus protocol. In addition, the designers find temporal logics intimidating and prefer state machine based specifications.

4 Conclusion

We have presented a tool set for modelling and verification of GALS applications. Tools such as SAL/PVS, Polis, Reactive Modules, and SMV capture both models of concurrency but significant semantic differences exist such as notions of acceptable programs and pre-emption mechanisms. Our work differs from most Statecharts based verification engines for similar reasons.

At present, we have developed static analysis[3] and refinement techniques[9] to ameliorate verification. These techniques are currently being incorporated in the tool. We are also exploring other techniques that might aid the designers and are conducting further case studies.

References

1. G. Berry. The constructive semantics of pure esterel. In *Book Draft, version 3*, July 1999.
2. G. Berry, S. Ramesh, and R. K. Shyamasundar. Communicating reactive processes. In *Twentieth Symposium on Principles of Programming Languages*, 1993.
3. Vinod Ganapathy and S. Ramesh. Slicing synchronous reactive programs. In *Synchronous Languages, Applications, and Programming*, Grenoble, France, April 2002.
4. Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.
5. Naveen Chandra R. Verification of communicating reactive state machines. M.Tech dissertation, Department of Computer Science and Engineering, IIT Bombay, 2002.
6. S. Ramesh. Communicating reactive state machines: Design, model and implementation. In *IFAC Workshop on Distributed Computer Control Systems*, September 1998.
7. S. Ramesh. Implementation of communicating reactive processes. *Parallel Computing*, 25(6):703–727, June 1999.
8. S. Ramesh and Chandrashekar M. Shetty. Impossibility of synchronization in the presence of preemption. *Parallel Processing Letters,*, 8(1):111–120, March 1998.
9. Sampada Sonalkar. Compositional verification of communicating reactive state machines. Master's thesis, Indian Institute of Technology Bombay, January 2004.
10. B. Vijayalakshmi. Verification of communicating reactive state machines. M.Tech dissertation, IIT Bombay, School of Information Technology, 2003.